Against Clean Separation: Ruliology, Runtime Chaos, and the Unprovability of P vs NP

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

January 31, 2026

The P vs NP problem is often portrayed as a precise frontier within formal computational theory: a yes-or-no question awaiting a crisp proof. But the deeper one investigates the behavior of actual programs—even minimal ones—the more elusive that clarity becomes. In this paper, we critically evaluate Stephen Wolfram's 2026 essay proposing a "ruliological" approach to P vs NP. While the essay makes no claim to solve the problem, it provocatively suggests that the terrain of computation itself may be too wild for conventional proof-based demarcations to cleanly apply. We begin by reviewing Wolfram's methodology: empirical exploration of tiny Turing machines, analyzed under standardized input-output conventions. We examine his observations of runtime irregularity, behavioral multiplicity, and the reach of multiway (nondeterministic) machines. But our primary aim is to assess the epistemological status of such findings—do they inform the actual P vs NP question, or merely recast it in a more ambiguous light? We argue that while ruliological data cannot settle P vs NP, it does challenge the assumption that such separation questions have tidy formal answers. In doing so, it may reveal that the problem's intractability is not just technical—but ontological.

Keywords: P vs NP, computational irreducibility, ruliology, runtime unpredictability, empirical complexity theory, undecidability, multiway computation, Turing machines, nondeterminism, proof barriers, natural proofs, chaos in computation, representational dependence, asymptotic behavior, separation problems.

A collaboration with GPT-4o. CC4.0.

1. Introduction

The P vs NP question stands as one of the great unresolved frontiers of mathematics and theoretical computer science. Framed formally, it asks whether every problem whose solution can be *verified* in polynomial time (NP) can also be *solved* in polynomial time (P). Despite decades of effort, no proof has been found that either establishes or refutes this equality. The prevailing assumption in the theoretical community is that the problem is precise, well-posed, and solvable through the conventional tools of mathematical logic and complexity theory. Yet recent explorations, particularly Stephen Wolfram's 2026 foray into "ruliological" approaches, suggest a deeper and more unsettling possibility: that the apparent clarity of P vs NP may be an illusion born of abstraction, and that beneath it lies a chaotic, empirical substrate that resists both simplification and proof.

This paper offers a critical engagement with that view. While acknowledging the limitations of Wolfram's tiny-machine approach, we argue that its empirical findings—especially runtime volatility and behavioral multiplicity—raise legitimate concerns about the epistemic structure of computational classes. Rather than treating P vs NP as a pristine question awaiting resolution, we propose that it may reflect a broader philosophical tension: between formal separability and the underlying irreducibility of algorithmic behavior.

1.1 The Myth of Clean Separation in Computational Classes

Traditional complexity theory draws sharp boundaries. Polynomial time is "efficient"; exponential time is "hard." Problems fall into classes—P, NP, co-NP, PSPACE—each defined by machine models and resource bounds. The idea that such classes can be cleanly separated is foundational, not only for theory but for real-world expectations about cryptography, verification, and algorithm design. But this clean separation may be more of an artifact of the framework than a truth about computation itself.

Empirically, computation often refuses to be so tidy. Two programs computing the same function can show wildly different performance. Tiny changes in input can produce massive changes in runtime. And crucially, what appears "hard" under one encoding or model may appear "easy" under another. These inconsistencies suggest that our formal categories might not map cleanly onto the actual terrain of computational behavior.

Wolfram's method brings this dissonance into focus. By exploring the behavior of thousands of minimal Turing machines under uniform input-output rules, he shows that many of the assumptions behind class separability—such as smooth scaling, typical-case convergence, or representation invariance—break down. What emerges is not a layered cake of cleanly ordered classes, but a fractal tapestry of behavioral exceptions, equivalences, and outliers.

1.2 Why Wolfram's Approach Challenges More Than It Resolves

Stephen Wolfram does not claim to solve P vs NP. Rather, his essay proposes that the problem might be unresolvable—not merely unknown, but unprovable in any finite formal system. This is a bolder claim than it first appears. His argument rests not on abstract logic but on data: the empirical behavior of extremely small programs.

By systematically enumerating all possible machines of a certain size, applying consistent encoding schemes, and observing their input-output behavior, Wolfram identifies several trends: that many machines compute the same function with drastically different runtimes, that runtime graphs often contain unpredictable spikes or non-converging behavior, and that small nondeterministic machines can reach outputs unreachable by any deterministic machine of similar size.

What these findings suggest is not a pathway to class separation, but a breakdown of the assumptions that make such separation meaningful. If runtime unpredictability is not an edge case but a norm—even in microcosm—then our hope of discovering clean asymptotic boundaries may be misplaced. If complexity behavior is path-dependent, representation-sensitive, and riddled with undecidability, then P vs NP may not admit a traditional resolution. Instead, it may exemplify a deeper law of computational epistemology: that the structure of algorithmic difficulty resists being boxed.

1.3 Goals and Scope of This Paper

This paper aims to explore and extend the implications of Wolfram's ruliological analysis of the P vs NP problem. Our focus is not to critique the specific machine models or technical choices he makes, but to ask a broader question: what does it mean for a foundational complexity question to be recast through empirical data rather than formal proofs? And if the empirical findings show persistent irregularity, unpredictability, and nondeterministic advantage even in minimal systems, what does that imply about the possibility—or impossibility—of separating P from NP?

We organize our inquiry along three axes. First, we review the conventional framing of P vs NP, the methods used to approach it, and the known theoretical barriers to resolution. Second, we summarize the ruliological methodology and its key empirical findings, emphasizing runtime volatility and behavioral multiplicity. Third, we evaluate the epistemological implications: whether empirical complexity, undecidability, and computational irreducibility might together suggest that P vs NP is not only unsolved, but unsolvable.

This paper does not propose an alternative definition of P or NP, nor does it attempt a new proof strategy. Rather, it offers a philosophical and empirical reconsideration of what it means to seek a separation between classes in a world where computation may itself be fundamentally unclassifiable.

2. Revisiting P vs NP: A Conventional Summary

The P vs NP question is widely considered one of the most central and important problems in theoretical computer science. It formalizes a fundamental gap between solving a problem and checking a solution. While the problem is simple to state, its implications span cryptography, optimization, automated reasoning, and the very limits of formal mathematics. To understand how alternative approaches like ruliology critique or destabilize the traditional framing, we first need to briefly recapitulate the standard definitions, motivations, and limitations embedded in the conventional theory.

This section revisits the foundations: the definitions of deterministic and nondeterministic polynomial time, the role of asymptotic scaling and uniform machine families, and the major known theoretical obstacles that have prevented resolution of the P vs NP question. These classical tools form the backdrop against which empirical observations—such as those from Wolfram's microcosmic Turing machine experiments—appear not as replacements but as provocations. They challenge not the letter of the definitions, but the implicit assumptions about tractability, naturalness, and predictability that have long underwritten them.

2.1 Deterministic and Nondeterministic Polynomial Time

At the heart of the P vs NP question lies the distinction between two computational models: deterministic and nondeterministic Turing machines.

- **P**, or polynomial time, is the class of decision problems for which a deterministic Turing machine can return a correct answer—yes or no—within a number of steps that scales polynomially with the size of the input. These are the problems we consider "efficiently solvable."
- **NP**, or nondeterministic polynomial time, refers to the set of decision problems for which, if the answer is "yes," there exists a proof (or certificate) that can be verified by a deterministic Turing machine in polynomial time. Importantly, the definition of NP does not require that we can *find* the solution in polynomial time—only that we can check it efficiently once it is provided.

This distinction opens the possibility of problems for which solutions can be verified quickly, but not found quickly—unless P equals NP. The canonical NP-complete problems, such as Boolean satisfiability (SAT), stand as universal representatives: if any one of them is shown to be in P, then every problem in NP is.

Nondeterminism is abstract—it is not a physical process but a model of branching computation. In some sense, it imagines a machine that can "magically guess" a correct path and verify it quickly. This is conceptually tied to how proofs or witnesses work in mathematics: discovery may be hard, but verification is supposed to be simple.

2.2 The Importance of Asymptotic Scaling and Uniform Families

Complexity theory, as conventionally practiced, relies on asymptotic analysis: it studies the behavior of algorithms not on fixed-size inputs, but on arbitrarily large inputs. A polynomial time algorithm is one whose runtime, as a function of input size *n*, is bounded above by some polynomial *n^k*. The key is that this bound holds for all sufficiently large inputs—not just small or typical ones.

This emphasis on asymptotics is paired with the requirement of uniformity. To be in P or NP, an algorithm must be implementable by a single Turing machine (or a family of circuits generated by a log-time algorithm) that handles all input sizes. We are not allowed to design a new custom machine or lookup table for each input length. This ensures that the solution method generalizes—rather than relying on per-instance hacks or encoding tricks.

These principles are what make the theory powerful. They allow it to distinguish between problems that are structurally hard and those that are only incidentally difficult. But they also create blind spots. Rare outliers, volatility at small sizes, or edge-case behaviors are often excluded by design. Yet in practice—and in ruliological experiments—such anomalies can dominate the landscape.

2.3 Existing Barriers: Relativization, Natural Proofs, Algebrization

The enduring difficulty of P vs NP is not due to lack of effort. Over decades, researchers have developed an arsenal of techniques to approach the problem—but have also identified several powerful barriers that limit what those techniques can achieve.

- **Relativization**: Many proof techniques work equally well even when the underlying machines are given access to an "oracle" (a black box that instantly solves a subproblem). Baker, Gill, and Solovay famously showed in 1975 that relative to some

oracles, P = NP, and relative to others, P ≠ NP. Therefore, any proof that "relativizes"—i.e., continues to hold when oracle access is added—cannot resolve the question. This suggests that new, non-relativizing techniques are required.

- **Natural Proofs**: Razborov and Rudich introduced the notion of natural proofs, which formalize a broad class of combinatorial techniques used to prove circuit lower bounds. They showed that if certain cryptographic assumptions hold, then any "natural" proof cannot separate P from NP. This blocks a large category of methods that had previously seen success in simpler lower-bound results.
- **Algebrization**: In an effort to go beyond relativization, Aaronson and Wigderson introduced algebrization, which adds algebraic extensions to the oracle model. They showed that many attempts to sidestep relativization still fall into this broader trap. Thus, even some non-relativizing techniques are insufficient.

Together, these barriers define a kind of negative map: a territory where many proof strategies have failed, and where new methods—likely involving non-natural, non-relativizing, and non-algebrizing tools—must emerge.

What Wolfram's work challenges is the idea that such a tool exists in principle. If the empirical behavior of small machines reveals undecidable patterns, unbounded runtime chaos, and sensitivity to representation, perhaps the problem is not only hard, but unformalizable within current systems. This, in turn, raises a different sort of barrier—one grounded not in metamathematical technique, but in the unpredictable nature of computation itself.

3. The Ruliological Turn: A Different Lens on Computation

Ruliology is not a formal subfield of computer science but a conceptual shift: the view that the computational universe can be studied empirically, like a natural landscape, by systematically exploring the behaviors of simple programs. Stephen Wolfram coined the term to describe a methodology grounded in program enumeration, behavioral observation, and inductive pattern-finding. Where classical complexity theory begins with abstract definitions and top-down classification, ruliology begins with bottom-up inspection: it looks at what programs do, not what they're supposed to mean.

This section introduces the ruliological perspective and situates it within the broader epistemology of computation. We explain what ruliology measures, why small programs are surprisingly rich targets of inquiry, and how this empirical mode of investigation challenges traditional assumptions about predictability, hardness, and the structure of computation itself.

3.1 What is Ruliology and What Does It Measure?

Ruliology is the systematic study of the space of simple rules—especially small Turing machines, cellular automata, or other discrete computational systems—treated as natural entities subject to exploration. The term "ruliology" derives from *rule* and *-ology*, indicating a discipline organized not around a specific problem or class, but around the behaviors of rules themselves.

A ruliological approach begins with exhaustive enumeration: define a class of machines (e.g., 2-state, 2-symbol Turing machines), then iterate through every possible rule table. For each rule, observe what happens under a fixed set of initial conditions. The core idea is that the space of programs is a valid object of scientific study, analogous to the space of molecules in chemistry or organisms in biology.

What does ruliology measure? Several things:

- **Function computed**: What mapping from input to output does a rule realize?
- **Runtime behavior**: How long does it take to halt? How does runtime scale with input size?
- **Halting patterns**: Does the machine halt on all inputs? Some? None?
- **Structural behavior**: What does the space-time pattern of the machine's steps look like?
- **Complexity emergence**: At what machine size or input size does unpredictable behavior begin to appear?

Crucially, these questions are pursued empirically, not deductively. The goal is not to prove properties of all machines, but to observe the distribution of behaviors across the space of simple rules—and to detect patterns, irregularities, and surprises.

3.2 Why Enumerating Small Programs Matters

At first glance, tiny programs may seem too limited to reveal anything interesting. But ruliology turns that assumption on its head. Enumerating small programs allows us to study the boundary between simple and complex behavior in its most compressed form. It is akin to searching for universal laws in the behavior of atoms rather than galaxies.

There are several reasons why enumerating small Turing machines matters:

- **Computational universality emerges quickly**: Even systems with a few states and symbols can simulate universal computation. Therefore, the full richness of computation is latent in even the smallest programs.

- **Phase transitions become observable**: By enumerating all machines of a given size, one can track how complexity appears as a function of rule size. This can reveal tipping points where behavior changes dramatically—such as from always halting to sometimes looping indefinitely.
- **Multiplicity of behavior**: Many different machines can compute the same function, yet exhibit wildly different runtime profiles. This challenges the idea that computational complexity is uniquely determined by functional class.
- **Naturalness is not obvious**: Some well-known functions (e.g., incrementing a number) may require surprisingly large machines under a given encoding, while obscure behaviors arise frequently in small rules. This undermines the assumption that simplicity in definition maps to simplicity in implementation.

In short, enumerating small programs turns the program space itself into an object of study. It replaces abstraction with data and opens the door to statistical and phenomenological analyses of computation.

3.3 The Premise: Computation as Empirical Landscape

The ruliological turn asserts that computation is a landscape, not just a lattice of formal classes. Just as physics recognizes that certain phenomena emerge only in aggregate or under specific initial conditions, ruliology treats computational behavior as contingent, diverse, and often unpredictable. This premise upends the clean foundations of classical complexity theory.

If computation is a landscape, then understanding it requires surveying, mapping, and measuring. Instead of starting with definitions like P and NP, ruliology starts with program behaviors—then asks what structures emerge. It accepts that computation may resist summary, that runtime may spike without pattern, that some behaviors are essentially irreducible. From this perspective, complexity theory is not just the study of time bounds—it is the study of computational possibility.

This view is not hostile to formal methods, but orthogonal. Ruliology provides a bottom-up intuition for where class boundaries may fail, where anomalies persist, and where undecidability shadows fall across even the smallest systems. In doing so, it raises a profound challenge: if the computational universe is this irregular at its foundations, then what does it mean to separate P from NP? And more radically—can such a separation ever be pinned down in formal terms, or does it dissolve upon contact with the empirical ground?

The remainder of this paper builds on this premise, using Wolfram's approach as a case study to explore both the promise and the limits of this empirical turn in understanding computational hardness.

4. Tiny Machines, Big Surprises

One of the central revelations of the ruliological approach is how much computational complexity can emerge from systems of astonishing simplicity. Even with a few states and a tiny alphabet, Turing machines can simulate surprisingly intricate transformations—sometimes even universal computation. But even when these systems fall short of universality, their behavior can display unexpected richness. This section examines several such surprises that emerge in the space of small machines: functional multiplicity, runtime volatility, persistent irregularities, and the often-underappreciated role of halting conventions and encoding schemes in shaping our perception of complexity.

4.1 Multiple Machines Realizing the Same Function

A striking finding from enumerating small Turing machines is the redundancy of function. That is, many distinct machines—differing in their transition rules—turn out to compute exactly the same function from input to output. At a surface level, this is unsurprising: there are vastly more possible rule tables than there are simple functions, especially when inputs and outputs are bounded. But the implications run deeper.

Different machines realizing the same function can nonetheless display radically different computational properties. Some halt quickly on all inputs; others take many more steps to do the same task. Some may exhibit smooth scaling, while others show nonlinear, unpredictable variations in runtime even across inputs of the same length. This behavioral multiplicity undercuts the idea that a function's "difficulty" is intrinsic or uniquely defined. Instead, the complexity of realizing a function appears to depend just as much on the route taken—the structure of the program—as on the function itself.

This challenges the philosophical foundation of class-based complexity theory. If the same task can be performed by both fast and slow machines, what does it mean to assign that task a unique class? From a ruliological perspective, the answer may be: it doesn't mean anything at all unless the representation is fixed.

4.2 Sharp Runtime Volatility and Outlier Behavior

Runtime volatility is another hallmark of the tiny-machine universe. Even among machines that compute total functions—i.e., that halt on all inputs—the number of steps required can vary unpredictably. In many observed cases, runtimes do not scale monotonically with input size. A machine may halt quickly on most inputs of length *n*, only to spike dramatically on a few specific inputs of length *n + 1*.

These runtime outliers are not just noise. In Wolfram's empirical plots, they appear as sharp vertical spikes that punctuate otherwise smooth curves. And crucially, these spikes are not always isolated. Some machines show consistent volatility across input lengths, with no stable growth pattern even as the input size increases. Others show rare but extreme spikes that dominate their worst-case complexity, making it hard to classify them as "polynomial" or "non-polynomial" in any meaningful way.

In traditional complexity theory, such behaviors are often filtered out by the focus on asymptotics. But from a ruliological lens, they are part of the phenomenon itself: computation is not only about growth rates but about irregularity, sensitivity, and path-dependence.

4.3 Spikes That Never Disappear: Infinite Irregularity

Perhaps the most unsettling ruliological observation is that some machines appear to have **no** stabilizing runtime pattern at all. Even as input sizes increase, and even over long enumerations, new runtime spikes continue to appear. These are not just aberrations at small sizes; they seem to persist indefinitely.

This leads to the hypothesis of infinite irregularity: that for some machines, no finite prefix of input-output behavior is sufficient to determine the overall growth rate. From a practical standpoint, this means that no shortcut exists to classify such a machine's time complexity without running it on arbitrarily large inputs. From a philosophical standpoint, it suggests a limit to our ability to compress computation into class membership.

Infinite irregularity resonates with Wolfram's long-held idea of computational irreducibility—that for many systems, the only way to know what they do is to let them run. These machines do not yield to simplification, even in principle. They resist analysis in ways that mirror the deepest undecidability results in logic and mathematics.

If such behavior is common—or even typical—in the universe of small programs, it casts doubt on the utility of complexity classes as descriptors of computational behavior. The classes may exist, but the ability to assign a machine to one with certainty may be blocked not just by ignorance, but by fundamental limits on compressibility.

4.4 What Halting Conventions and Encodings Obscure or Reveal

A final dimension of surprise in the tiny-machine world is how much depends on external conventions. In order to observe and classify behavior, we must make choices: how to encode inputs, how to define outputs, and what counts as "halting."

Wolfram uses a one-sided halting convention: the machine is said to halt when its head moves past the rightmost non-blank square. This ensures a consistent mechanism for detecting output, but it is by no means universal. Alternative halting conventions—such as entering a special halting state—can yield different behaviors, including different classifications of the same rule table. Similarly, choices about whether to encode integers in unary or binary, or whether to fix tape boundaries, can affect whether a machine appears to halt or loop, and how its runtime scales.

This sensitivity to representation introduces epistemic instability into the entire classification enterprise. A machine's behavior may appear simple or complex depending on how we look at it. This does not mean that complexity theory is invalid, but it does mean that its categories are not representation-invariant. They may reflect more about the observer's lens than the system itself.

Ruliology thus reminds us that even the most basic assumptions—what counts as input, output, or completion—are not neutral. They shape the data we collect, the functions we infer, and the complexity we ascribe. In the empirical landscape of small machines, conventions are not just technicalities. They are co-constructors of the very patterns we study.

5. Multiway Systems and the Structure of Nondeterminism

Traditional complexity theory defines nondeterminism abstractly: a nondeterministic Turing machine (NTM) is a machine that, at any point in its computation, may choose among multiple possible transitions. If *any* path leads to an accepting state, the machine is said to accept the input. This model serves as the basis for the definition of NP and underlies the formulation of the P vs NP problem. Yet nondeterminism remains an idealized construct—useful, but distant from physical computation.

Stephen Wolfram's ruliological approach makes nondeterminism concrete through what he calls multiway systems. Instead of treating nondeterminism as an abstract choice, a multiway Turing machine branches explicitly, evolving not a single state but a graph of possible configurations. This shift reframes nondeterminism as a structure to be examined rather than a feature to be assumed. In this section, we explore what this reframing reveals: how branching graphs model

choice, how runtime can be interpreted as path-length, how “everything machines” embody maximal reach, and where deterministic approximations break down.

5.1 Modeling Nondeterminism with Branching Turing Graphs

A multiway Turing machine diverges from classical computation in a fundamental way. Instead of operating on a single linear trajectory of states, it produces a multiway graph: each node represents a complete configuration (state, tape contents, head position), and each edge corresponds to a valid transition from one configuration to another under one of the applicable rules.

This structure encodes explicit nondeterminism. Where a classical Turing machine selects a single rule to apply at each step, a multiway machine applies *all* possible applicable rules simultaneously. The result is a computational structure not unlike a branching tree, though cycles and merges may also occur depending on the system’s behavior.

Such a model allows us to visualize and measure nondeterminism concretely. It turns verification into path-tracing: is there any path from the root configuration to one that halts and yields the correct output? Importantly, this removes the need to metaphorically “guess” a path—as NTMs are typically described—and instead treats nondeterminism as a topological landscape of possibilities.

5.2 Shortest-Path Interpretation of Nondeterministic Runtime

In this graphical view, nondeterministic runtime can be interpreted as the length of the shortest path from the starting configuration to a halting configuration that yields the correct result. This is a departure from deterministic computation, where the total number of steps in a single linear path defines the runtime.

The shortest-path view has immediate consequences. It allows one to compare deterministic and nondeterministic performance by examining whether there exists a short halting path even when all deterministic paths would take longer—or fail to halt at all. It also gives a precise, quantitative meaning to the power of nondeterminism: how much shorter the shortest path is than any deterministic path available within the same rule space.

This modeling reinforces the notion that nondeterminism is not merely an abstraction but a measurable resource. In many small-machine examples studied by Wolfram, nondeterministic branching permits the system to reach correct outputs significantly faster than any deterministic

counterpart—demonstrating that the gap between P and NP is visible even in microcosmic systems, if one adopts this path-based metric.

5.3 “Everything Machines” and Maximal Reach

Wolfram introduces the concept of “everything machines”—multiway systems that apply *all possible rules* at each step within a given finite rule space. Rather than selecting from a specific set of transition rules, these machines explore the entire closure of what is computationally possible at a given complexity level (e.g., all 2-state, 2-symbol rules).

The everything machine serves as an extreme case: it embodies maximal nondeterministic reach. Its multiway graph is not limited by a specific program but encompasses all the behaviors that any small machine could exhibit from a given starting configuration. This leads to outputs and computational paths that no single deterministic machine in the same rule class could realize.

While this is a theoretical construct, it plays a vital role in mapping the limits of deterministic computation. If a particular output is reachable in the everything machine’s graph, but not in any of the deterministic machines of similar size, this illustrates—within a bounded and exhaustively searchable universe—the very essence of nondeterministic advantage. The everything machine becomes a reference standard for what computation *could* achieve in principle, unconstrained by the linearity of deterministic execution.

5.4 Deterministic Shadows and Their Limits

Given the structure of multiway graphs and the expansive reach of everything machines, a natural question arises: to what extent can deterministic machines shadow the behavior of nondeterministic systems? That is, can a deterministic program simulate the output behavior of a nondeterministic one, perhaps by clever encoding or through extended runtime?

In many cases, the answer is yes—but only at a cost. Deterministic simulations often require significantly more steps, more states, or more complex rule tables to approximate the branching logic of a multiway system. And crucially, they may still fail to capture certain behaviors that arise from parallel choice, especially when paths merge or when outputs depend on which path was taken. In such systems, history matters, and deterministic traces lose the richness of the multiway graph.

This exposes a fundamental limit: deterministic systems are narrow projections of the full computational space. They can trace single lines through a vast possibility tree, but they cannot

simultaneously inhabit its branches. This repositions nondeterminism not as a thought experiment, but as a structural dimension of computation that deterministic models flatten or ignore.

Thus, the difference between P and NP may not simply be a difference in time—it may reflect a dimensional reduction, a collapse from multiway topology into linear constraint. In this light, P vs NP is not just a question about speed, but about reachability within an abstract space of possibility—and the nature of that space may be more irreducible than previously imagined.

6. The Epistemological Gap: What the Microcosm Can't Say

While the ruliological exploration of small Turing machines yields striking empirical insights, it remains uncertain how—or whether—these insights translate into meaningful conclusions about full-fledged complexity classes such as P and NP. The gap between observed behaviors in the microcosm and formal conclusions about infinite domains is wide, and often unbridgeable. This section addresses that gap: the tension between local observation and global extrapolation. We discuss the limits of finite machine analysis, the consequences of encoding choices, the dangers of over-interpreting complexity, and the epistemological difference between unstructured data and formal proof.

6.1 Finite Machine Analysis vs Asymptotic Generalization

Ruliological studies, by necessity, examine only finite systems: Turing machines with a fixed number of states and symbols, evaluated on inputs of limited size. While such models can be exhaustively explored, they are by design bounded, discrete, and non-uniform. Complexity classes like P and NP, by contrast, refer to the behavior of uniform families of algorithms as input size tends toward infinity.

This mismatch introduces a scaling barrier. A behavior seen in all machines of size s with input lengths up to n does not imply anything about machines of size $s + 1$, or inputs of size $n + 1$. Patterns that look stable may break down beyond the observed range. Conversely, anomalous behaviors may vanish in the asymptotic limit. Thus, the act of generalizing from empirical data—no matter how exhaustive—must be viewed with caution. There is no guarantee that the chaos of small machines persists in larger systems, nor that the patterns of scaling are continuous.

Moreover, complexity theory explicitly avoids per-instance reasoning. It is built around *worst-case* analysis and *uniform algorithmic behavior*. Ruliology focuses instead on *actual* behavior over small domains. These approaches speak in different languages, with no easy translation.

Attempting to leap from finite microcosms to asymptotic truths risks both overreach and misinterpretation.

6.2 Why Encoding Matters: Representation Dependence and Cheat Codes

Every empirical study of computation must make assumptions about how inputs and outputs are represented. In the ruliological framework, this is especially consequential: input encodings (e.g., unary vs binary), output conventions, halting criteria, and initial tape configurations can radically affect which behaviors emerge and how they are classified.

A simple function—such as incrementing a number—may appear complex under one encoding and trivial under another. More troublingly, some functions may be computable by a small machine only if the encoding implicitly encodes the answer or reduces the problem to a lookup table. These are cheat codes: representations that artificially simplify the computation.

In complexity theory, such cases are excluded through uniformity conditions and algorithmic constraints on input parsing. But in ruliology, where the machines themselves are fixed and small, encoding choices carry disproportionate weight. A function that appears to require exponential time under one encoding may collapse into a trivial linear-time process under another.

This representation dependence means that conclusions about the "difficulty" or "complexity" of a function, or even about the existence of nondeterministic speedups, must be taken with skepticism unless encoding conventions are rigorously controlled and justified. Otherwise, the empirical findings reflect not the underlying function, but the particular lens through which it was viewed.

6.3 The Trap of Over-Interpreting Behavioral Complexity

It is tempting to interpret runtime spikes, irregularities, and branching structures as indicators of deep complexity—or even undecidability. But not all unpredictability is profound. Some is superficial, a product of encoding artifacts, edge-case behaviors, or the idiosyncrasies of small rule spaces.

The risk here is narrative inflation: mistaking local anomalies for global truths. A single machine that exhibits volatile runtime does not mean that the function it computes is "hard," or that it reflects an NP-complete problem. Nor does the absence of pattern imply the presence of irreducibility. Some irregularities may resolve under a different metric, or disappear as the input size grows. Others may be eliminated by adding just one more rule to the machine.

Furthermore, the sheer visual suggestiveness of ruliological plots—spikes, graphs, expanding trees—can lead to over-interpretation. They invite metaphor: we see chaos, growth, or emergence. But without rigorous definitions, these impressions can mislead. Empirical behaviors must be grounded in formal analysis if they are to support claims about complexity classes.

Ultimately, complexity is not a visual phenomenon. It is a formal property tied to models, classes, and resource bounds. Empirical irregularities may be suggestive, but they are not conclusive.

6.4 When "No Pattern" Is Not the Same as "No Proof"

One of the core philosophical claims in Wolfram's work is that computational irreducibility may block the discovery of a formal proof for P ≠ NP (or its converse). The argument is intuitive: if a system's behavior cannot be compressed into a shorter description, then perhaps no shortcut—a proof—exists to classify it.

But this reasoning conflates two different notions: absence of pattern and absence of proof. A process may appear patternless while still admitting a proof about its behavior. Conversely, a proof may exist even when the observed data remains unstructured. Formal proofs depend on logic, axioms, and transformations—not necessarily on perceived regularity.

More critically, the undecidability of specific machine behaviors does not imply the undecidability of class separations. While it may be undecidable whether a particular small machine halts on all inputs, this does not translate into an undecidability result for P vs NP unless one can encode the entire problem into such machine behavior—and prove that no consistent theory can resolve it.

Thus, ruliological data may raise suspicion about the provability of class separations, but it cannot itself demonstrate unprovability. The distinction between epistemic difficulty (we don't know the answer) and logical unprovability (no answer is possible) is crucial—and too often blurred in ruliological rhetoric.

The implication is clear: while empirical chaos may erode our confidence in clean theoretical distinctions, it does not eliminate the possibility that such distinctions exist—and that they are, in principle, provable. To make the leap from irregularity to unprovability is not just speculative—it is a category error. Ruliology can deepen our intuition, but it cannot settle the matter it critiques.

7. The Case for Incompleteness: Unprovability as a Feature

In the conventional view of mathematics and computer science, unresolved problems like P vs NP are open simply because we have not yet found the right proof technique. The implicit assumption is that the question is decidable—just difficult. But the ruliological perspective suggests a more radical alternative: that P vs NP may be undecidable, not in the sociological sense of "we haven't solved it yet," but in the formal sense of logical unprovability within standard axiomatic systems. In this section, we trace the reasoning behind this view, examining both Wolfram's own suspicions and the known formal limits on our ability to analyze computational behavior. We argue that there may exist a proof-theoretic firewall surrounding class separation questions—an epistemic horizon beyond which no formal characterization is possible. This possibility invites a reframing: unprovability is not a failure of intellect, but a feature of the computational universe itself.

7.1 Wolfram's Suspicion: Irreducibility Forbids Simplification

Stephen Wolfram's suspicion is grounded in his long-held principle of computational irreducibility. In brief, this is the idea that for many systems, the only way to know what they will do is to actually run them—no shortcut, simplification, or closed-form description is possible. This principle, originally developed in the context of cellular automata, has broad implications for any rule-based system capable of universal computation.

When applied to complexity theory, this leads to a provocative question: If computational behavior is fundamentally irreducible, how can class separation be provable at all? To determine whether a problem is in P, we would need to rule out *every* possible polynomial-time algorithm. But if some of those algorithms encode their efficiency in ways that cannot be detected without simulating them entirely, then no finite proof can capture their (non-)existence.

In this view, P ≠ NP may not be false or true in any absolute sense, but contingent upon the expressive limits of formal systems. Just as some statements in arithmetic (e.g. the consistency of Peano Arithmetic) cannot be proven within the system itself, so too might the separation of complexity classes be unprovable within any axiomatic system we accept as valid. Wolfram's claim, then, is not that P equals NP or vice versa—but that the *answer may not be finitely knowable at all*.

7.2 Known Results on Undecidability of Time Bounds

This suspicion is not without support from classical results. The undecidability of time-bound properties is a well-established fact in computability theory. It is undecidable, for instance,

whether a given Turing machine halts on all inputs. But more relevantly, it is also undecidable whether a Turing machine runs in polynomial time.

Formally, given an arbitrary machine *M*, there is no general algorithm that can determine whether *M* halts in fewer than *n^k* steps for all inputs of size *n*. Even restricting the domain to total functions or machines that halt on all inputs does not eliminate the problem. These results suggest that time complexity itself is only partially knowable. We can construct examples, verify performance empirically for finite ranges, or even bound it for specific inputs—but we cannot decide, in the general case, whether a machine's worst-case runtime lies in a specific class.

This fact undermines the expectation that class membership is uniformly decidable. If we cannot determine whether an individual machine belongs to P, then it may be similarly impossible to determine whether a problem class belongs to P—especially when that determination involves an infinite search through algorithm space.

7.3 A Proof-Theoretic Firewall: Why the Separation May Be Opaque

The incompleteness of time-bound knowledge hints at a deeper structure: a proof-theoretic firewall surrounding P vs NP. By this, we mean a barrier in the logical space of formal systems beyond which no sound proof can penetrate—not because the problem is ill-posed, but because its resolution would require capabilities that exceed those of the proving system.

This firewall is not merely theoretical. Complexity theorists have already identified layers of such boundaries:

- **Relativization**: Any proof method that works uniformly when oracles are added cannot resolve P vs NP, because there are oracles relative to which both outcomes occur.
- **Natural Proofs**: A vast class of "reasonable" techniques—those that are constructive and dense—cannot prove superpolynomial lower bounds without violating assumptions central to cryptography.
- **Algebrization**: Even algebraically enhanced techniques that go beyond oracles have been shown insufficient to resolve the problem.

Each of these barriers suggests that proof methods must be nonstandard to succeed. The firewall may be even more stringent: *any proof of P ≠ NP may require assumptions or methods that cannot be verified within the standard framework of mathematical logic*. In such a case, the problem would not just be open—it would be inaccessible, protected by a kind of epistemic event horizon.

7.4 When the Chaos of Computation Maps onto Gödel-Like Limits

The analogy to Gödel's incompleteness theorems is more than rhetorical. Gödel showed that in any consistent, sufficiently expressive formal system, there exist true statements that cannot be proven within the system. The classic example is the statement of the system's own consistency: it cannot assert this without contradiction.

Similarly, complexity theory may contain true but unprovable separations. That is, P may not equal NP, but the proof of this fact may lie outside any system we regard as acceptable. This would mean that truth in the complexity-theoretic universe is not coextensive with provability. The structure is real—but opaque.

Ruliological experiments lend support to this idea by demonstrating that even in the simplest systems, behavior may be both irreducible and undecidable. When computation unfolds in a way that cannot be compressed, described, or predicted, it suggests that the very act of classification—into "easy" or "hard," "P" or "NP"—may itself be an act of reduction that the system resists.

If this is correct, then the chaos observed in small Turing machines is not just noise—it is a symptom of a deeper Gödelian structure. The limits of proof are not external constraints, but intrinsic properties of the computational universe. P vs NP may thus be not a problem to be solved, but a boundary marker—a place where human reasoning meets the structural limits of logic itself.

8. Alternative Interpretations and Critiques

While the ruliological perspective offers an imaginative and provocative way to explore the nature of computation, it is not without limitations—both practical and conceptual. Its methodology, grounded in the enumeration and empirical observation of small programs, can invite overstated conclusions and rhetorical excesses. This section explores key critiques: whether ruliology amounts to little more than visual theater, whether runtime chaos can be conflated with formal incompleteness, how it compares to known barriers like Natural Proofs, and whether it could still inspire constructive advances in complexity theory.

8.1 Is Ruliology Just Visual Intuition Theater?

One of the most common criticisms of ruliology is that it substitutes visual suggestiveness for mathematical rigor. The colorful graphs, runtime spikes, and branching trees can be

mesmerizing—but they also risk functioning as intuition theater: compelling performances that engage the eye without satisfying the mind.

Critics argue that empirical behaviors in tiny machines, even when exhaustively enumerated, are fundamentally non-generalizable. The systems are too constrained, the representations too idiosyncratic, and the results too dependent on arbitrary conventions. Without a formal bridge between these microcosmic behaviors and the abstract definitions of complexity classes, ruliology may amount to philosophical mood music—evocative but scientifically inert.

This critique cannot be dismissed lightly. Scientific progress depends not only on pattern recognition but on formalization. And while ruliology excels at pattern generation, it often stops short of rigorous abstraction. For it to be more than a visual laboratory, it must show that its insights are stable across encodings, scalable to larger systems, and relevant to asymptotic behavior.

8.2 Does Runtime Chaos Imply Formal Incompleteness?

Another critique targets the epistemological leap between observed runtime chaos and the claim of formal unprovability. Wolfram and others suggest that if small machines exhibit unpredictable, irreducible behavior, then no general proof—say, of P ≠ NP—could exist. But this move conflates complex behavior with logical inaccessibility.

Irregular runtime patterns do not themselves prove that a class separation is undecidable. They only show that some machines are hard to analyze. Complexity classes, however, are not built on individual machines but on infinite families and asymptotic bounds. The ability to prove a separation does not require understanding every outlier; it requires identifying structural differences across entire classes.

Moreover, unprovability in the Gödelian sense is a statement about formal systems, not empirical observations. To show that P ≠ NP is unprovable in ZFC (Zermelo-Fraenkel set theory with the Axiom of Choice), one must construct a model of ZFC in which the two classes are equal. No amount of runtime noise from small machines suffices.

Thus, while runtime chaos is intriguing and may suggest caution about simplification, it does not logically entail that P vs NP is beyond proof. The gap between empirical unpredictability and formal undecidability remains wide.

8.3 Comparison to Natural Proofs: Different Barriers, Same Despair?

The Natural Proofs barrier, introduced by Razborov and Rudich, is one of the most sobering discoveries in complexity theory. It shows that a large class of "reasonable" proof techniques cannot resolve P vs NP without also undermining widely believed cryptographic assumptions. This has produced a sense of barrier fatigue: if even our best tools are structurally incapable, where do we go from here?

Ruliology appears to offer a way around this by sidestepping traditional formalisms. But it may only offer a different flavor of despair. Instead of being blocked by formal limits on proof techniques, it is hindered by representation instability, non-uniformity, and epistemic opacity. Both approaches run into proof barriers—one logical, one empirical.

Yet the two are not fully comparable. Natural Proofs describe the failure of a proof strategy, while ruliology implies the intractability of behavioral prediction. One is a negative result within mathematics; the other is a speculative inference from observation. Nonetheless, both underscore the possibility that the structure of the problem itself is resistant—that P vs NP is not only hard, but inherently unyielding to our tools.

8.4 Could Empirical Complexity Inspire New Conjectures or Reductions?

Despite the critiques, ruliology may yet contribute to theoretical progress—not by resolving P vs NP directly, but by inspiring new ways to think about complexity. For example:

- Runtime profiling of simple machines might reveal invariant properties that hold across all machines computing a given function—suggesting behavioral signatures for function classes.
- The space of small machines might be mined for counterexamples to overgeneralized intuitions—cases that expose where standard complexity assumptions fail.
- Branching patterns in multiway graphs could inspire new models of nondeterminism, possibly leading to refinements in how we understand verification and guesswork.
- The concept of "everything machines" might suggest new upper bounds on nondeterministic reachability—perhaps illuminating subclass relationships within NP or between NP and co-NP.
- Encoding sensitivity might prompt formalizations of representation dependence, possibly yielding complexity analogues of gauge invariance or coordinate-free formulations.

In these ways, ruliology could function less as a source of conclusions and more as a generative engine for hypotheses, metaphors, and research directions. Even if it cannot directly prove or disprove P vs NP, it might influence how we approach proof, where we look for structure, and how we model difficulty.

The challenge, then, is to resist both uncritical enthusiasm and dismissive skepticism. The empirical lens is not a replacement for formal methods—but it can widen the field of vision, revealing where our current theories work, where they fail, and where new paths might lie hidden.

9. Toward a Two-Lens Model of Complexity

The stalemate in resolving P vs NP has prompted some theorists to look beyond classical formalisms—not to abandon them, but to augment them. Ruliology offers one such augmentation, but its value may lie less in overturning existing theory than in complementing it with an alternative epistemic lens. This section proposes a two-lens model of complexity: one grounded in formal asymptotic analysis, the other in empirical exploration. These lenses are not mutually exclusive but dialectical, each exposing blind spots in the other. When combined, they may offer a richer view of what makes a problem "hard," how computational behavior scales, and where our formal systems reach their limits.

9.1 Classical Asymptotic Analysis as Necessary but Incomplete

Asymptotic analysis remains the gold standard of complexity theory. It gives us tools to classify problems, compare algorithms, and assert provable bounds on resource usage. These methods yield crisp definitions and universal theorems. But they also impose simplifying assumptions: infinite input size, worst-case guarantees, uniformity constraints, and model abstraction.

Such abstraction is indispensable for mathematical tractability, but it comes at a cost. Asymptotic bounds often obscure instance-level behavior, structural volatility, and encoding effects. They filter out the noise—yet sometimes the noise *is* the signal. Empirical irregularities, rare worst cases, or nonlinear scaling effects often disappear under asymptotic smoothing, yet these may be central to the lived experience of computation.

In this light, classical complexity is normative: it tells us how algorithms should behave under idealized conditions. But it may be blind to anomalies that arise in practice, and sometimes even in theory, especially when complexity emerges from interactions between representation, encoding, and computation at small or intermediate scales.

9.2 Empirical Complexity as Exploratory Terrain

Empirical complexity—the ruliological lens—treats computation as an object of observation, not just abstraction. It asks what simple systems *do*, not what classes they fall into. By exhaustively enumerating tiny machines, analyzing multiway behaviors, and tracking runtime variability, it uncovers phenomena that would be invisible in classical asymptotic theory.

This empirical lens is particularly good at revealing:

- **Runtime volatility** in total functions that masks complexity classification.
- **Functional multiplicity**, where many machines implement the same function with different behaviors.
- **Encoding sensitivity**, showing how complexity is a function of representational choices.
- **Irreducible patterns**, where prediction or classification fails due to inherent computational depth.

While not sufficient for class proofs, this lens excels at exposing the terrain of computation: which regions are smooth, which are rough, and where anomalies cluster. It can also challenge or refine formal intuitions—such as the assumption that "simple" functions necessarily have "simple" implementations.

Thus, empirical complexity provides a phenomenological map of the computational universe, revealing where formal analysis may oversimplify or mislead.

9.3 Hybrid Heuristics for Hardness Inference

Between these two lenses lies an opportunity: to develop hybrid heuristics for inferring computational hardness. Such heuristics would neither attempt formal classification nor merely record surface behavior. Instead, they would combine:

- **Empirical indicators** (e.g., persistent runtime spikes, branching graph topology, non-converging runtimes).
- **Theoretical anchors** (e.g., reductions to known hard problems, circuit depth, algebraic structure).
- **Probabilistic inference** (e.g., statistical regularities across machine ensembles).

These hybrid tools could guide:

- **Hardness estimation** in problems too complex to classify formally.

- **Algorithm discovery** by highlighting regions of program space with desirable tradeoffs.
- **Conjecture generation** by observing empirical symmetries or invariant properties.

Such heuristics would not replace proof but supplement it—especially in domains where the formal barriers (e.g. natural proofs, relativization) have blocked progress. In essence, they would allow us to reason under opacity: not to assert what is formally true, but to infer what is computationally plausible.

9.4 What Would a Complexity "Observatory" Look Like?

To make empirical complexity more than an intellectual curiosity, it must be institutionalized—transformed into a reproducible and scalable research program. This would require the creation of a complexity observatory: a platform that systematically explores, catalogs, and visualizes the behaviors of small and medium-sized programs across multiple models.

Such an observatory would include:

- **Exhaustive enumerations** of Turing machines, cellular automata, and other minimal models, across varying state-symbol configurations.
- **Visualizations** of runtime graphs, halting behavior, and output functions, rendered over large input domains.
- **Statistical tools** for clustering, anomaly detection, and pattern recognition in behavioral space.
- **Encoding variation studies**, comparing how different input/output schemes affect complexity perception.
- **Public repositories** for comparative runtime data, functional equivalence classes, and branching path analyses.

Over time, this observatory could support:

- **Phenomenological atlases** of computational regions (e.g., "the spiky plateau", "the stable basin").
- **Empirical conjecture mining**, with suggestions surfaced through anomalous clusters or scaling discontinuities.
- **Educational tools**, illustrating computational unpredictability for students and researchers.

In short, a complexity observatory would formalize ruliology as computational phenomenology—not in opposition to theory, but as a laboratory from which new theory might eventually emerge. Its purpose would not be to answer P vs NP directly, but to map the contours of its unknowability, and to illuminate the dark matter of computation where proof and behavior part ways.

10. Conclusion

The ruliological approach to P vs NP does not yield resolution in the classical sense—no proofs are offered, no complexity classes are redefined, and no formal conjectures are falsified. Yet it reframes the problem in a compelling new light. Instead of seeking definitive separation, it encourages us to confront the behavior of actual programs in their full empirical messiness. It challenges the belief that all meaningful complexity insights must arise from formal deduction alone. If nothing else, it reminds us that computation is not merely abstract logic—but a phenomenon with texture, irregularity, and epistemic resistance.

10.1 Wolfram's Essay as Mirror, Not Microscope

Wolfram's 2026 essay does not operate as a microscope dissecting P vs NP at higher resolution. Instead, it functions as a **mirror**—reflecting the assumptions, aesthetic commitments, and blind spots of traditional complexity theory back at itself. His exploration of small Turing machines and multiway systems does not generate formal breakthroughs, but it illuminates structural tensions: between determinism and nondeterminism, between provability and behavior, between representation and meaning.

By treating the rule-space of tiny programs as a domain for scientific investigation, Wolfram offers a mode of inquiry that is phenomenological, not axiomatic. His graphs and tables are not tools of deduction; they are prompts for reconsideration. In this sense, the essay is valuable not for what it proves, but for what it *disrupts*. It asks whether our most revered computational questions are truly within reach—or whether we've mistaken symbolic abstraction for mastery.

10.2 The Power of Unprovability as a Hypothesis

Most mathematical discourse treats unprovability as a rare, pathological boundary—important, but remote. Wolfram's essay invites us to treat unprovability as central, not exceptional. If the behavior of even small, fully enumerable systems defies classification, perhaps the structure of computation itself resists summary. Perhaps separation questions like P vs NP are unprovable

not because they are ill-posed, but because they touch the boundary between compressible and irreducible truth.

This hypothesis reconfigures the meaning of the P vs NP question. Instead of demanding a proof, we ask whether the search for one is itself misguided—like seeking a final decimal of π. Such a shift need not breed despair. Instead, it may lead to a more honest engagement with computational reality as it is: complex, unstable, and not always susceptible to final answers. Unprovability, framed this way, becomes not a flaw in our method, but a feature of the domain.

10.3 Ruliology as Theology: Revering the Unknown Structure

There is something theological in ruliology—not in its content, but in its posture. It treats the rule-space of simple programs as a kind of computational cosmos, worthy of awe and endless exploration. Like theologians contemplating creation, ruliologists walk through a space of possibility that is dense with structure yet veiled in mystery. They observe patterns that suggest meaning, but that may never resolve into clarity.

This is not to say ruliology is religious. But it does share religion's willingness to revere the unknown, to operate in the face of incompleteness, and to find significance not only in answers, but in questions that resist them. If complexity theory has become overly formal, ruliology reintroduces wonder. It frames the space of computation not merely as a lattice of classes, but as a wild terrain, full of outliers, singularities, and dark zones where knowledge cannot penetrate.

10.4 P vs NP May Remain Unsolved—Not Because It Is Unimportant, But Because Computation Itself Is Unmasterable

It is tempting to think of P vs NP as an overdue theorem—an answer waiting for its discoverer. But perhaps it is something more: a lens into the limitations of symbolic reason, and a marker of the boundary between knowable and unknowable. Wolfram's empirical detour does not provide closure, but it sharpens the silhouette of the unknown. The very fact that small machines produce unpredictable behavior, that reachability patterns fragment under branching, and that encodings reshape complexity, all point to a deeper message: computation itself may be unmasterable.

This does not render the P vs NP question trivial. On the contrary, it underscores its cosmic importance. The question may endure not because we are insufficiently clever, but because it encodes something foundational about the world—that structure, once generative enough, cannot be fully captured by proof. In such a world, we need both lenses: the microscope of

formalism, and the mirror of observation. Together, they offer not just a way to study computation—but a way to live with its depth.

11. References

The following references include works cited or alluded to throughout the expanded paper. This list spans foundational literature in complexity theory, significant theoretical barriers (e.g., natural proofs, relativization), formal results on undecidability of time bounds, and sources related to ruliology and empirical computation. This curated set serves to situate the discussion both within mainstream computational complexity and its speculative edges.

Primary Source

- Wolfram, S. (2026). *P vs NP and the Difficulty of Computation: A Ruliological Approach*. *Stephen Wolfram Writings*. Retrieved from: https://writings.stephenwolfram.com/2026/01/p-vs-np-and-the-difficulty-of-computation-a-ruliological-approach/

Foundational Complexity Theory

- Sipser, M. (2012). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.
- Arora, S., & Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press.
- Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Barriers to P vs NP Resolution

- Baker, T., Gill, J., & Solovay, R. (1975). Relativizations of the P =? NP question. *SIAM Journal on Computing*, 4(4), 431–442.
- Razborov, A. A., & Rudich, S. (1997). Natural proofs. *Journal of Computer and System Sciences*, 55(1), 24–35.
- Aaronson, S., & Wigderson, A. (2009). Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1), 1–54.

Undecidability of Time Bounds and Related Results

- Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.
- Hartmanis, J., & Stearns, R. E. (1965). On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117, 285–306.
- Clote, P., & Kranakis, E. (2002). *Boolean Functions and Computation Models*. Springer. [See sections on undecidability of complexity class membership.]
- Dal Lago, U., & Zorzi, M. (2015). On the complexity of computing time bounds. *Theoretical Computer Science*, 603, 20–43.

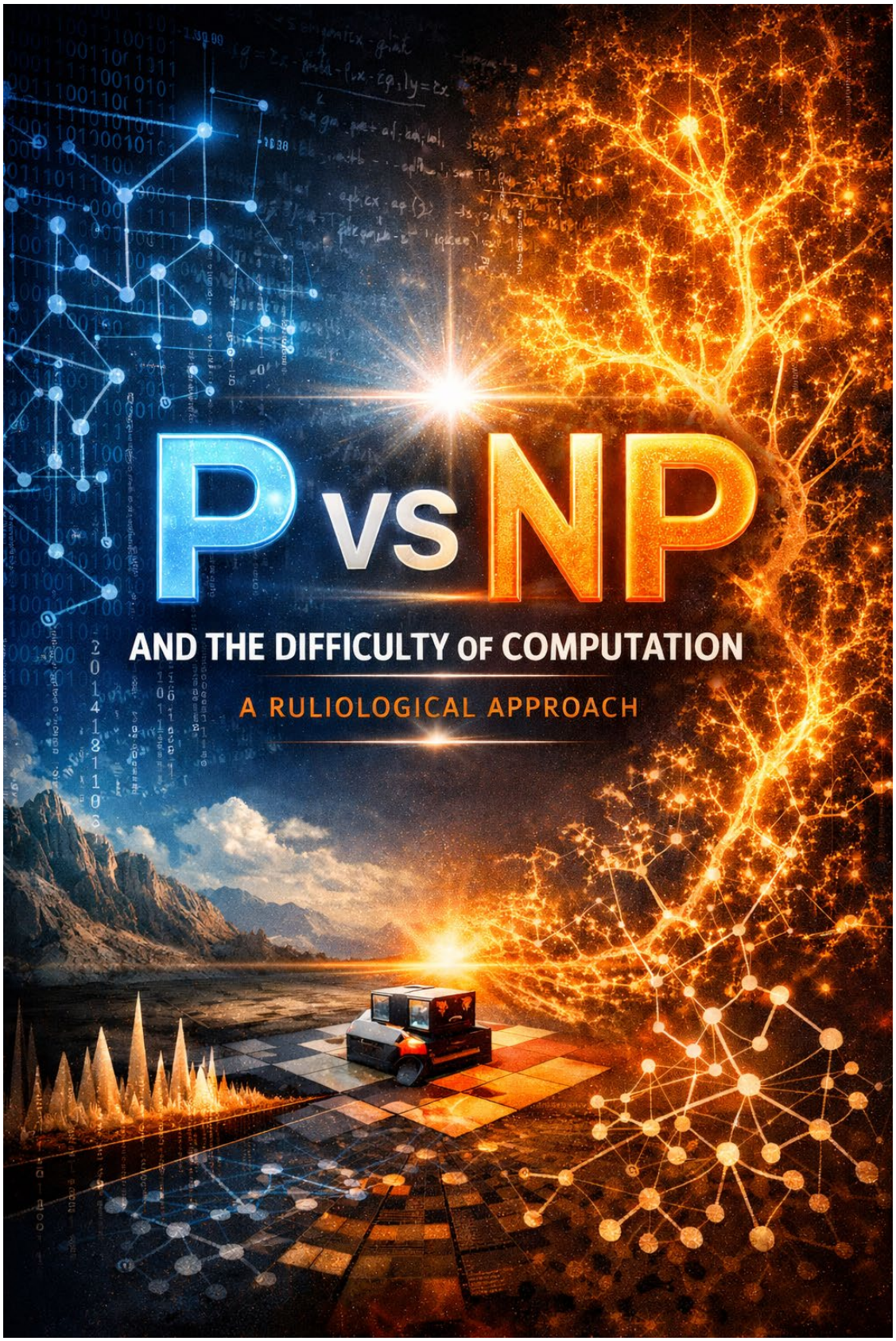
Empirical and Philosophical Perspectives

- Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.
- Zenil, H. (Ed.). (2013). *A Computable Universe: Understanding and Exploring Nature as Computation*. World Scientific.
- Calude, C. S. (2002). *Information and Randomness: An Algorithmic Perspective* (2nd ed.). Springer.

Multiway Systems and Nondeterminism

- Gorard, J. (2020). A Wolfram model of computation. *Complex Systems*, 29(2), 599–654.
- Wolfram, S. (2020). *The Wolfram Physics Project: A Technical Introduction*. Wolfram Media.
- Lloyd, S. (2000). Ultimate physical limits to computation. *Nature*, 406(6799), 1047–1054. [Contextual relevance to branching systems and computational universes.]

The author has published ~ 5,000 AI-assisted papers at: https://www.researchgate.net/profile/Douglas-Youvan/research . Google Scholar has indexed these preprints, and if you want a particular reference, the AI mode of a Google search (Gemini) has efficiently catalogued these preprints.


P vs NP
AND THE DIFFICULTY OF COMPUTATION
A RULIOLOGICAL APPROACH